AMENDMENTS TO THE SPECIFICATION

Please replace the paragraph beginning on at page 1, line 6 with the following:

The present application claims the benefit of~~priority from~~ United States Provisional Patent Application Nos. 60/399,635 entitled "Data Dispersion and Mirroring Method with Fast Dual Erasure Correction and Multi-Dimensional Capabilities" filed on July ~~28~~29, 2002 and 60/460,545 entitled "Composite Data Protection Method for Micro Level Data" filed April ~~3~~4, 2003.

Please replace the paragraph beginning on page 2, line 24 with the following:

Consider the CCITT 16-bit polynomial code. The generator polynomial is g(x) = 1 + x^{5}~~5~~ + x^{12}~~12~~ + x^{16}~~16~~. It has a Hamming distance d = 4 giving it the following capabilities: 1) It can detect all combinations of 3 random bit errors over its length of 65536 bits, d = t + 1; 2) It can detect all odd bit errors; and 3) It can detect all burst errors of up to 16 bits length.

Please replace the paragraph beginning on page 3, line 15 with the following:

In the book Algebraic Coding Theory, (Elwyn Belekamp 1968) the author published that the polynomial whose roots are the finite field elements of order p is called the cyclotomic polynomial, denoted by $Q^{(p)}(x)$. He noted that the factors of $Q^{(17)}(x)$ are g1(x) = 1 + x^{3}~~3~~ + x^{4}[[4]] + x^{5}~~5~~ + x^{8}~~8~~ and g2(x) = 1 + x + x^{2}~~2~~ + x^{4}[[4]] + x^{6}~~6~~ + x^{7}~~7~~ + x^{8}~~8~~ and that each polynomial could correct two random errors in a block of 17 binary digits.

Please replace the paragraph beginning on page 5, line 6 with the following:

The prior art includes some solutions based on the dispersal of data with the recovery via dual erasure control coding. In the context of small disk arrays this is commonly referred to as method RAID level 6. The number of disk drives clustered in these arrays typically varies from as few as four to six to as many as nine or seventeen. Other erasure control applications may be called information dispersal and reconstruction, erasure or loss resilient coding or in specific context forward error correction coding, FEC.

Please replace the paragraph beginning on page 5, line 8 with the following:

United States Patents 5,128,810 (Halford) and 5,283,791 (Halford), in the name of the present inventor, show dual drive error correction method using a data bit dispersal with erasure control technique. This method requires a parity side-band drive and a significant amount of computation due to the dual dependent ECC codes (odd parity and irreducible polynomial). This approach is relatively cumbersome and slow if implemented in software. Also the ~~rectangular~~ code data array is rectangular M * (M-1) which limits data size flexibility.

Please replace the paragraph beginning at page 11, line 10 with the following:

It is noted by Berlekamp that the factors of $Q^{(17)}$ (x) are a pair of degree 8 irreducible self-reciprocal polynomials capable of performing error correction. These factors are g1(x) = 1 + x^{3}~~3~~ + x^{4}[[4]] + x^{5}~~5~~ + x^{8}~~8~~ and g2(x) = 1 + x + x^{2}~~2~~ + x^{4}[[4]] + x^{6}~~6~~ + x^{7}~~7~~ + x^{8}~~8~~ . While each of these irreducible polynomials work equally well the patent application will nominally refer to g1(x). The ECC byte is determined by dividing the data byte by one of the irreducible polynomials g1(x) = 1 + x^{3}~~3~~ + x^{4}[[4]] + x^{5}~~5~~ + x^{8}~~8~~ and multiplying by x^{8}~~8~~. The register labeled 200 is first cleared to all zeros prior to the operation. Each data byte is divided by the polynomial g1(x) = 1 + x^{3}~~3~~ + x^{4}[[4]] + x^{5}~~5~~ + x^{8}~~8~~ and multiplied by x^{8}~~8~~ by inputting the data byte serially into the x^{8}~~8~~th exclusive-or position 201, of the 8-bit feedback shift register labeled 200. The residual data, after all 8-bits are input becomes the ECC value. The example in Figure 2 is for data byte 9C hexadecimal. The resultant ECC after 8 shifts is 80 hexadecimal. Field 7 of Table 2 and Table 4 is a listing for all ECC values. Once the table is developed it can be used for software or firmware implementations without the need for the regeneration of ECCs. The summation of the contributing data terms for each ECC term results in a set of equations translating data to ECC. For hardware applications the ECC can also be evaluated more efficiently via these boolean logic equations also shown in Figure 2D item 210. In a similar process the equations can be generated that translate ECC bits to data bits. These are noted in Figure 2D item 220.

Please replace the paragraph beginning at page 13, line 14 with the following:

Given a data byte [Di] = d0d1d2d3d4d5d6d7 construct an ECC byte [Ei] = e0e1e2e3e4e5e6e7 via the self-reciprocal irreducible polynomial g1(x) = 1 + x^{3}~~3~~ + x^{4}[[4]] + x^{5}~~5~~ + x^{8}~~8~~ or g2(x) = 1 + x + x^{2}~~2~~ + x^{4}[[4]] + x^{6}~~6~~ + x^{7}~~7~~ + x^{8}~~8~~. The codeword for each data byte becomes [CWi] = [Di] [Ei] = d0d1d2d3d4d5d6d7e0e1e2e3e4e5e6e7 where i is the ith information byte in the datagram. With ts = the data byte and vu = the ECC byte and st, su, sv, tu, tv, and uv, are row couplets mirroring the data byte [Di] = st and wx, wy, wz, xy, xz and yz are column couplets also mirroring the data byte [Di] = st. [Di] is found by boolean evaluation or table look-up method given any row or column couplet. It can be said that all couplets mirror all other couplets. Via the inherent capability of the ECC all codewords [CWi] = [Di] [Ei] have self error detecting capability. Via the inherent capability of the ECC all codewords [CWi] = [Di] [Ei] have 2-bit random error correcting capability.

Please replace the paragraphs beginning at page 14, line 20 through line 29 with the following paragraphs:

Consider the codeword in array form. If we know wx = d0 d4 e0 e4 d1 d5 e1 e5 we can find yz = d2 d6 e2 e6 d3 d7 e3 e7 by combination and substitution using the ECC equations of Figure 2 item 210. Similarly all ten sets of eight equations can be developed. The ECC is found by dividing the data byte by the irreducible symmetric polynomial g1(x) = 1 + x$^{\underline{3}}$~~3~~ + x$^{\underline{4}}$[[4]] + x$^{\underline{5}}$~~5~~ + x$^{\underline{8}}$~~8~~. The circuit depicted in Figure 2A is a feedback shift register with exclusive-or gates in positions 3, 4, 5 and 8. A faster method is also shown that evaluates all 8 bits in parallel.

The ECC values for all 256 possible data bytes can be computed once and saved in a list or table for look-up. They are summarized as follows for columns and rows.

For G1(x) = 1 + x$\underline{3}$~~3~~ + x$^{\underline{4}}$[[4]] + x$^{\underline{5}}$~~5~~ + x$^{\underline{8}}$~~8~~

Please replace the paragraph beginning at page 15, line 19 with the following:

Data can be evaluated from column couplets (for generator polynomial g1(x) = 1 + x$^{\underline{3}}$~~3~~ + x$^{\underline{4}}$[[4]] + x$^{\underline{5}}$~~5~~ + x$^{\underline{8}}$~~8~~) :

Please replace the paragraph beginning at page 16, line 16 with the following:

Data can be evaluated from row couplets (for generator polynomial g1(x) = 1 + x$^{\underline{3}}$~~3~~ + x$^{\underline{4}}$[[4]] + x$^{\underline{5}}$~~5~~ + x$^{\underline{8}}$~~8~~) :

d0 = **d0st,su,sv** **or** (d4 + d5 + d7 + e3)**tu** **or** (d4 + d6 + d7 + e5 + e6)**tv** **or** (e2 + e3 + e4 + e5)**uv**

d1 = **d1st,su,sv** **or** (d4 + d5 + d6 + d7+ e1)**tu** **or** (d5 + d7 + e6 + e7)**tv** **or** (e0 + e3 + e4 + e5 + e6)**uv**

d2 = **d2st,su,sv** **or** (d5 + d6 + d7 + e2) **tu** **or** (d4 + d6 + e4 + e5)**tv** **or** (e1 + e4 + e5 + e6 + e7) **uv**

d3 = **d3 st,su,sv** **or** (d6 + d7 + e0 + e3) **tu** **or** (d5 + d6 + e4 + e5 + e7)**tv** **or** (e3 + e4 + e6 + e7) **uv**

d4 = **d4st** **or** (d1 + d2 + e1 + e2)**su** **or** (d0 + d1 + d3 + e4)**sv** **or** **d4tu,tv** **or** (e0~~e~~ + e2 + e3 + e7)**uv**

d5 = **d5st** **or** (d2 + d3 + e0 + e2 + e3)**su** **or** (d0 + d1 + d2 + e4 + e7)**sv** **or** **d4tu, tv** **or** (e0~~e~~ + e1 + e2 + e5)**uv**

d6 = **d6st** **or** (d0 + d1 + e1 + e3)**su** **or** (d0 + d1 + d2 + d3 + e5)**sv** **or** **d4tu,tv** **or** (e0~~e~~ + e1 + e2 + e3 + e6)**uv**

d7 = **d7st** **or** (d0 + d1 + d3 + e0 + e1)**su** **or** (d0 + d2 + e4 + e6)**sv** **or** **d4tu ,tv** **or** (e1 + e2 + e3 + e4 + e7)**uv**

Please replace the paragraph beginning at page 17, line 13 with the following:

For G2(x) = 1 + x + x^{2}~~2~~ + x^{4}[[4]] + x^{6}~~6~~ + x^{7}~~7~~ + x^{8}~~8~~

Please replace the paragraph beginning at page 19, line 1 with the following:

Data can be evaluated from row couplets (for generator polynomial g2(x) = 1 + x + x2 + x4 + x6 + x7 + x8) :

d0 = d0**st,su,sv or** (d4 + d5 + d6 + d7 + e2)**tu or** (d4 + d7 + e4 + e5)**tv or** (e0 + e2 + e5 + e7)**uv**

d1 = d1**st,su,sv or** (d5 + d6 + d7 + e3)**tu or** (d4 + d6 + d7 + e6 + e7)**tv or** (e1 + e2 + e3 + e5 + e6 + e7)**uv**

d2 = d2**st,su,sv or** (d4 + d5 + d6 + e0 + e3) **tu or** (d4 + d5 + e4 + e5 + e7)**tv or** (e3 + e4 + e5 + e6) **uv**

d3 = d3 **st,su,sv or** (d4 + d6 + e0 + e2 + e3) **tu or** (d4 + d5 + d6 + d7 + e4 + e6)**tv or** (e0 + e4 + e5 + e6 + e7) **uv**

d4 = d4**st or** (d0 + d1 + e2 + e3)**su or** (d1 + d2 + d3 + e5)**sv or d4tu,tv or** (e0~~e~~ + e1 + e2 + e6)**uv**

d5 = d5**st or** (d2 + d3 + e1 + e2)**su or** (d1 + d3 + e4 + e7)**sv or d4tu, tv or** (e0~~e~~ + e1 + e2 + e3 + e7)**uv**

d6 = d6**st or** (d0 + d1 + d3 + e0)**su or** (d0 + d1 + e4 + e5 + e7)**sv or d4tu,tv or** (e0~~e~~ + e1 + e3 + e4 + e5 + e7)**uv**

d7 = d7**st or** (d0 + d2 + e0 + e1 + e2 + e3)**su or** (d0 + d1 + d2 + d3 + e4)**sv or d4tu ,tv or** (e1 + e4 + e6 + e7)**uv**

Please replace the paragraph beginning at page 18, line 4 with the following:

Data can be evaluated from column couplets (for generator polynomial g2(x) = 1 + x + x^2~~2~~ + x^4[[4]] + x^6~~6~~ + x^7~~7~~ + x^8~~8~~) :

Please replace the paragraph beginning at page 19, line 1 with the following:

Data can be evaluated from row couplets (for generator polynomial g2(x) = 1 + x + x^2~~2~~ + x^4[[4]] + x^6~~6~~ + x^7~~7~~ + x^8~~8~~) :

Please replace the paragraph beginning at page 21, line 12 with the following:

It is an object of the invention that degree 8 polynomials other than those noted could be used to obtain very similar results. It is an object of the invention that hardware implementations make use of the boolean equations noted earlier in the summary for encoding, decoding and reconstruction. Different equations with very similar capabilities exist when using the polynomial g2(x) = $1 + x + x^{2}$~~2~~ $+ x^{4}$[[4]] $+ x^{6}$~~6~~ $+ x^{7}$~~7~~ $+ x^{8}$~~8~~ or other capable polynomials. It is also noted that the codewords can be arranged in different configurations without changing the overall capabilities.

Please replace the paragraph beginning at page 22, line 17 with the following:

Consider a dual-element data set [st] conventionally mirrored by two additional data sets [uv] and [qr]. A system with this much redundancy must anticipate more than two potential failures before corrective reconstruction can be scheduled. Consider the effects of three and four failures. Clearly five failures within a six element system results in a catastrophic loss of data. For conventional triplicate redundancy there are two combinations of three failures out of the 20 possible combinations that would result in a catastrophic loss of data. They are [suq] and [tvr]. Likewise there are 6 combinations of four failures out of the 15 possible combinations that would also result in a catastrophic loss of data. They are [tvqr], [tuvr], [suqr], [suvq], [stvr] and [stuq]. Simple triple redundancy appears to have substantial mathematical deficiencies with eight failing combinations.

Please replace the paragraph beginning at page 22, line 27 with the following:

The invention can disperse rows q, r, s, t, u and v and recover all but three combinations of 1, 2, 3 and 4 element failures. Random dual bit error correction and data validation is possible as with the inventions four element dispersal method. It is also an object of the invention to permit dispersal of 4-bit column elements m, n, o and p.

Please delete the paragraphs beginning at page 23, line 21 ("A second method ...") and ending with page 24, line 9 ("...the dispersal path in error.")

Please replace the paragraph beginning at page 25, line 3 with the following:

~~And~~A further object ~~too~~ of the invention is that the codeword packet can be developed over time and stored in dispersed locations and on disparate media. For instance the data bytes could be stored on disk drives for fast random access performance and the ECC bytes could be on tape. Another strategy is to have the data bytes on disk and the complete codeword distributed across 4 tape cartridges. Reading any two tape cartridges successfully would recover data bytes lost on tape or disk. Backup to tape could occur overnight when the disk files see potentially less activity and snapshots of data are easier.

Please replace the paragraph beginning at page 25, line 14 with the following:

While the present invention ~~disclosure~~ consistently shows a data byte distributed equally across all four [[4]] elements of an array it is not the intention of the invention to restrict higher level hardware, software or firmware from distributing records, sectors, files or data bases onto individual~~individuals~~ elements of the array. The invention simply makes it possible to distribute each byte over the array for security concerns.

Please replace the paragraph beginning at page 25, line 19 with the following:

Another object of the invention is that data can be dispersed to four devices then during read back the four [[4]] data segments are assessed as to which devices need recovery and whether the error correction or the erasure recovery method should be used. Also it may be advantageous that device groups can fluctuate between two and four units as resiliency requirements change. In a virtualized system all devices can be non-arrayed and the application or file system decide on the resiliency level and choose to disperse data to two, three or four devices. Also it may be advantageous to migrate data first to two devices, one containing data and the other ECC bytes then later migrate data to "s" and "t" devices and ECC to "u" and "v" devices.

Please replace the paragraph beginning at page 30, line 4 with the following:

The fastest method for hardware logic designs are the column and row erasure equations listed in this section of the disclosure titled Data evaluated from column couplets using generator polynomial $g1(x) = 1 + x^{3}$~~3~~ $+ x^{4}$[[4]] $+ x^{5}$~~5~~ $+ x^{8}$~~8~~ . These equations can also be implemented via specialized scalar and vector processor programmed instructions. Since we know wx = d0 d4 e0 e4 d1 d5 e1 e5 we can find yz = d2 d6 e2 e6 d3 d7 e3 e7 by combination and substitution using the ECC equations of Figure 2 item 210. [The plus sign, + , denotes the exclusive-or boolean logic operator, d are binary data bits and e are binary ECC bits]. To evaluate couplet wx we use only the wx terms of the column evaluation logic terms listed above.

Please replace the paragraph beginning at page 33, line 26 with the following:

Once the data packet exits the CRC Checker, item 705, it is buffered in the Data Buffer In logic block, item ~~200~~700. If no data packet CRC error is detected at the node input the buffered data packet can be moved into the node, item 760. Here again it is prudent to reverify the CRC as the data packet can possibly be corrupted in the buffering process. The data passes from data buffer, item 700 to the node, item 760, through the second CRC Checker, item 730. If a CRC error is detected the Message Control In logic block, item 715, is again signaled to reply to the transmitting node with a NACK signal. A CRC error occurring at either CRC Checker will flag the data packet to be in error at the node, item 760. If no CRC errors have been detected the Message Control In logic block, item 715 replies~~replys~~ to the transmitting node with an Acknowledge signal.

Please replace the paragraph beginning at page 33, line 14 with the following:

The explanation block ~~790~~795 details the encoding of ACK and NACK signals. Effectively there is no special encoding at this level, an active ACK line translates as an ACK and an active NACK line translates as a NACK. Setting both ACK and NACK is not applicable and disallowed.

Please replace the paragraph beginning at page 35, line 29 with the following:

Figure 10 shows that byte 0 of data packet [D] 1000 is received with bit 5 picked; '81' hexadecimal has become 'A1' hexadecimal. The ECC value for 'A1' is found by evaluation using the equations of Figure 2C item 210 or from Table 2 as 'F8' hexadecimal. The ECC packet [E] 1010 byte 0 read is '57' hexadecimal. Subtracting '57' hexadecimal from 'F8' hexadecimal using the binary exclusive-or operator we get the ECC syndrome 'AF' hexadecimal. From Table 5 which uses the generator polynomial g1(x) = 1 + $x^{\underline{3}}$~~3~~ + $x^{\underline{4}}$[[4]] + $x^{\underline{5}}$~~5~~ + $x^{\underline{8}}$~~8~~ we obtain the error pattern of d5 and e5. The table actually used by hardware, software or firmware would likely produce the binary error pattern of '20' hexadecimal for correction of just the data byte and ignoring the ECC byte. The error pattern is exclusive-or'ed with the data by toggling bit 5 back to the correct value of '81' hexadecimal.

Please replace the paragraph beginning at page 38, line 3 with the following:

Step 1285 sends ACK and NACK signals requesting a retry of the previous data packet then advances to step ~~1220~~1225.

Please replace the paragraph beginning at page 40, line 9 with the following:

Referring now to Figure 19, thereshown is flowchart of process for encoding codeword arrays. After starting at block 300, a Message Byte Count is set to zero in block 305. Next, at block 310, a Next Data Byte [D] is retrieved. Then, at block 315, Table 1 is addressed with Data Byte [D] and a codeword is loaded. An optional step of generating separate Cyclic Redundancy Check (CRC) for each Data Byte may be performed at block 320. Then, at block 325, data buffers are aggregated for dispersion of the packet across multiple channels (here 4). The Message Byte Count is then incremented at block 330. The process then determines at block 335 whether an entire message has been encoded. If no, the process cycles back to block 310. If yes, the process determines whether a CRC is to be performed at block 340. This process is optional. If ~~the~~ no ~~such~~ CRC is to be performed, the process stops at block 345. If a CRC is to be performed, the process continues on at block 350 by encoding CRC bytes in the same manner as the data bytes in block 350. Table 1 is then addressed with the CRC byte and loaded with a codeword array at block 355. Again, data buffers are aggregated for dispersion of the codeword array elements across multiple channels in block 360. The CRC count is then incremented in block 365 and the process determines if it has reached the end of the CRC process. If not, the process loops back to block 350. If so, the process ends at block 375.